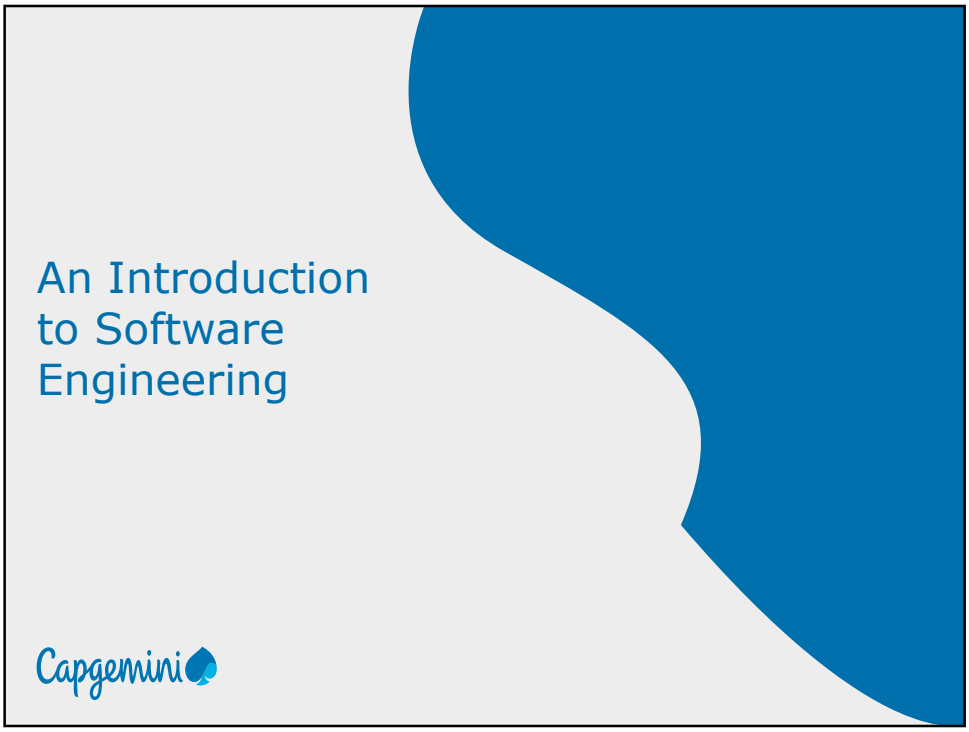
Instructor Notes:



Notes:

Instructor Notes:

Document History

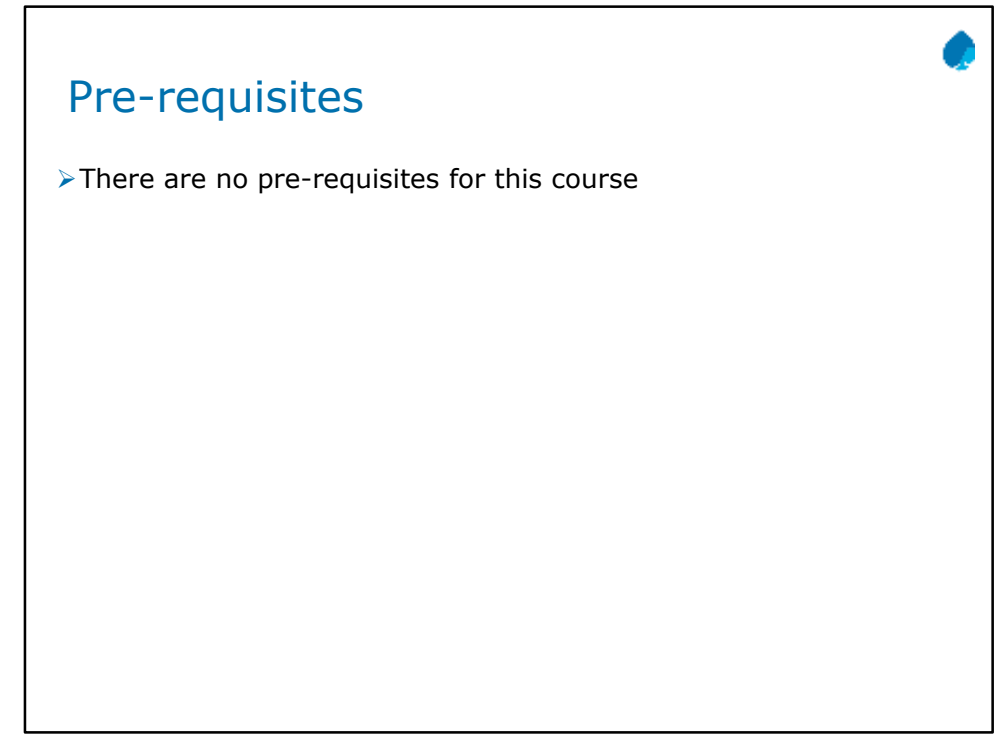
Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
21-May-2013	0.1D		Latha S	
2-Sep-2013	1.0		Latha S	Incorporated Review comments
July-2016	1.1		Anjulata Tembhare	Refinements

Instructor Notes:

Course Goals and Non Goals

- Course Goals
  - To provide an overview of software engineering
- Course Non Goals
  - This course is only an introductory course to Software engineering and does not intend to delve into details on activities involved in phases like requirements, high level design etc.

Instructor Notes:



Instructor Notes:

Intended Audience

- New entrants to the organization (Fresher's batches)

Instructor Notes:

Tell the participants that we are just briefly touching these topics for their general awareness purpose. Some of these topics have a separate training program of their own.

Objective

- To Understand the following :
  - What is Software Engineering (SE)
  - Common life cycle models
  - Phases in SE
  - Familiarizing Requirements Phase
  - Familiarizing Design Phase
  - Familiarizing Construction Phase
  - Familiarizing Testing and acceptance Phase
  - Review and Configuration Management Process

Instructor Notes:

Instructor Notes:

Overview of the Session

- What is Software Engineering?
- Software Development Life Cycle
- Software development Models
- Life cycle selection

Instructor Notes:

Software Engineering – What

- A systematic, disciplined and measurable approach towards development, operation and maintenance of a software
- Concerned with creating and maintaining software applications by applying technologies and practices from
  - Computer science,
  - Project management,
  - Engineering,
  - Application domains etc..
- It is broad term covering not only the technical aspects of building software , but also other factors like team management, schedule, budget and resource management etc..

Typical other formal definitions of software engineering are

"an engineering discipline that is concerned with all aspects of software production"
"the establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines“

Traditional engineers use science to construct “real” artifacts and software engineers use mathematics, science to construct “abstract” artifacts

In layman terms it is application of engineering towards development of a software

Instructor Notes:

Software Development Life Cycle (SDLC)

- Also known as software development process or Systems development life cycle
- A set of processes, standards and tools used to develop, alter software in a optimal manner
- Starts when a product is conceived and ends when the product is no longer available or is effective to use
- Composed of phases , where each phase is dependent on the previous phase's result
- Each phase is a limited period of time starting with a definite set of data and having a definite set of results

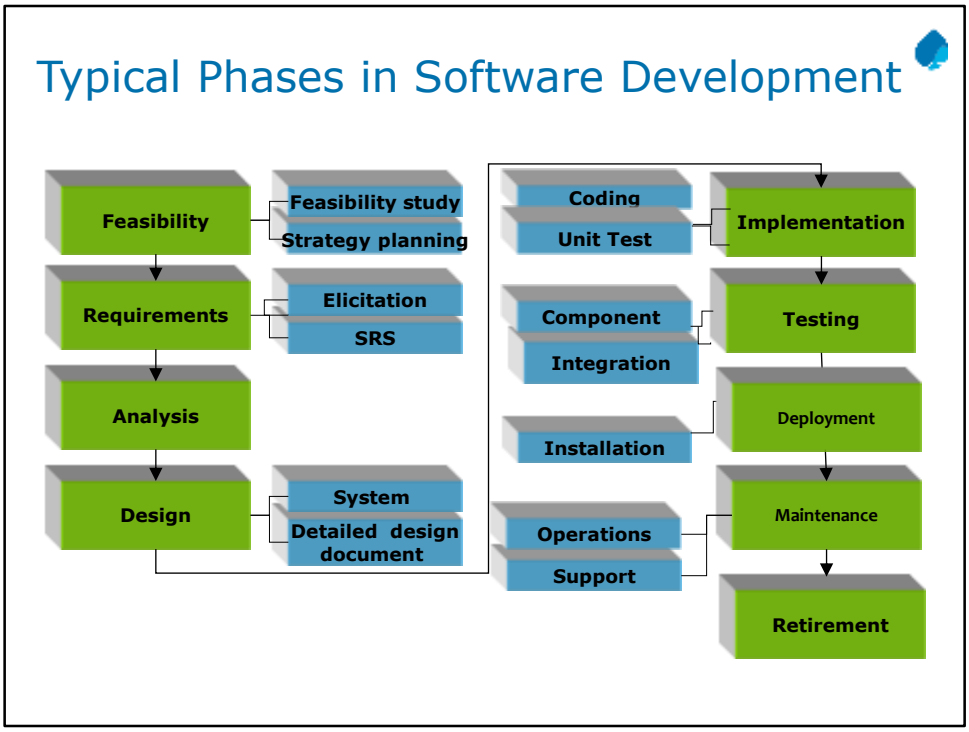
Also known as **systems development life cycle (SDLC)**, or **software development process**, or **Software Development Life Cycle**

It is a process of creating or altering information systems using various models and methodologies The SDLC aims to produce a high quality system that meets or exceeds customer expectations, reaches completion within times and cost estimates, works effectively and efficiently

The SDLC framework provides a sequence of activities for system design and development . It consists of a set of steps or phases in which each phase of the SDLC uses the results of the previous one.

.

Instructor Notes:



Activities during phases
Requirements: establish the customer's needs
System Design: develop the system's structure
Detailed Design: develop module structures
Implementation: code or otherwise
Testing: check what's been developed
Installation: bring the system into production
Maintenance: correct, adapt, improve

Instructor Notes:

SDLC Models

- A life cycle model covers the entire lifetime of a software – from birth of an idea to phase out
- More than one possible life cycle models can be adopted
- The type of SDLC model is defined by the way it links the phases.
- Every life cycle focusses its phase towards a goal and has a definite milestone
- Some of the common developmental models defined are
  - Waterfall /Enhanced Waterfall
  - V – model
  - Evolutionary Prototyping (aka Incremental)
  - Throw-away Prototyping (aka Rapid))
  - Incremental
- Following models are typically used in the organisations Iterative , V-model , Agile and semi waterfall

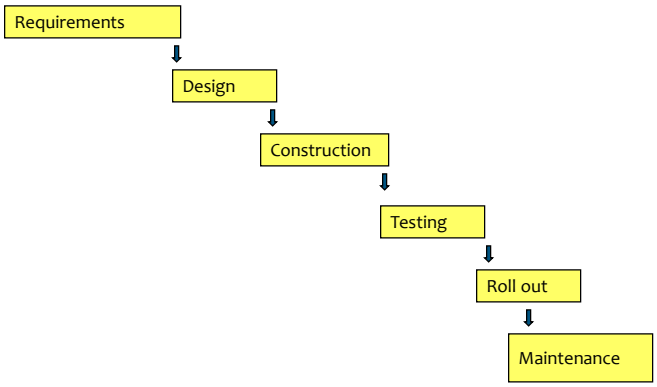
A (software/system) *lifecycle model* is a description of the sequence of activities
carried out in an SE project, and the relative order of these activities.

Provides a generic framework for various activities to be done for the lifetime of the s/w - design, develop and maintain

More than 1 model can be chosen or models can be changed between releases of the s/w

Instructor Notes:

Software Development Models- Waterfall



Waterfall processes

The waterfall model is a sequential software development model in which development is seen as flowing steadily downwards (like a waterfall) through the phases of requirements analysis, design, implementation, testing (validation), integration, and maintenance.

To follow the waterfall model, one proceeds from one phase to the next in a purely sequential manner. For example, one first completes "requirements specification" — they set in stone the requirements of the software.When and only when the requirements are fully completed, one proceeds to design. The software in question is designed and a "blueprint" is drawn for implementers (coders) to follow — this design should be a plan for implementing the requirements given. When and only when the design is fully completed, an implementation of that design is made by coders. Towards the later stages of this implementation phase, disparate software components produced by different teams are integrated.

After the implementation and integration phases are complete, the software product is tested and debugged; any faults introduced in earlier phases are removed here. Then the software product is installed, and later maintained to introduce new functionality and remove bugs.

Thus the waterfall model maintains that one should move to a phase only when its preceding phase is completed and perfected. Phases of development in the waterfall model are thus discrete, and there is no jumping back and forth or overlap between them.

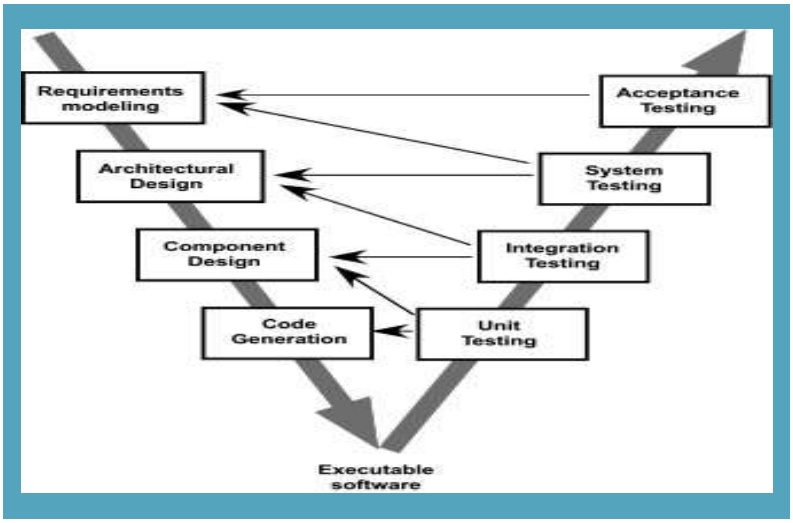
Typically Waterfall model is chosen when requirements , development environment and technology are well known and is more or less permanent

Instructor Notes:

Instructor Note :

Since both testing and development are done in parallel , there is always a discussion whether it is testing or developmental model

Software Development Models – V Model



It is also known as Verification and Validation model .

This is a SDLC model which emphasizes the verification and validation of the product. The quality assurance activities are performed in the each phase of Software Testing Life Cycle phase. Based on the requirement document both development team and testing team start their activities in parallel . The developer team started working on the design and after completion on design start actual implementation . The testing team in parallel starts working on test planning, test case writing, test scripting..

Main focus in V mode l is proactive defect tracking and avoiding the downward flow of defects . However though this model helps in planning for Verification and Validation in early stages of product development , it cannot handle the following

- concurrent events
- dynamic changes in requirements
- risk analysis and feedback to the previous iteration

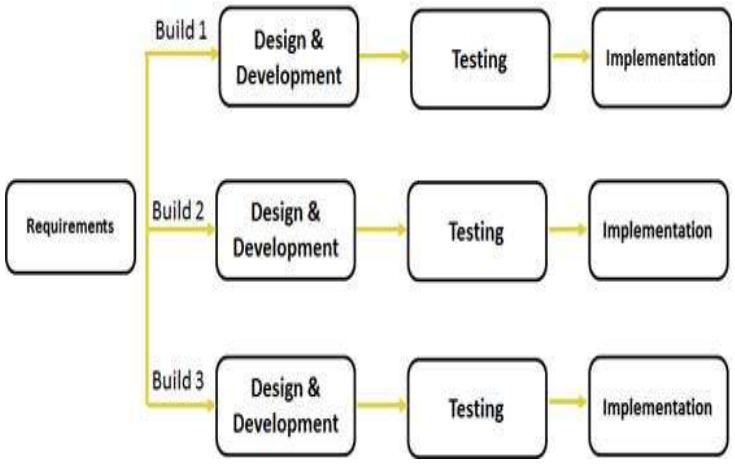
V-Model is ideal for projects where requirements are well known upfront , needs very high level of reliability and is of small size

Instructor Notes:

Instructor Notes

Explain that there are many types of incremental model UP, RUP etc .

Software Development Models – Iterative and Incremental



The iterative and incremental model (IID) method of s/w development is to build the s/w in a incremental way . Every increment will involve analyzing requirements , design , code , test , deploy iteratively adding a little more to the product until the whole product is finished.

The basic idea behind iterative enhancement is to develop a system incrementally, allowing the development team to take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system. Learning comes from both the development and use of the system, where possible.
Key steps in the process were to start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving sequence of versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added.

The goal for the design and implementation of any iteration is to be simple, straightforward, and modular, supporting redesign at that stage

Note : Fixing defects identified is NOT an iteration. Every iteration has a concrete deliverable and undergoes all the phases . Requirement prioritization helps in deciding the deliverables in each iteration . For example we may deliver web interface in first iteration and mobile interface in second iteration

This model is suitable for projects having evolving requirements , need to get to market early . Lengthy schedule and new technologies .

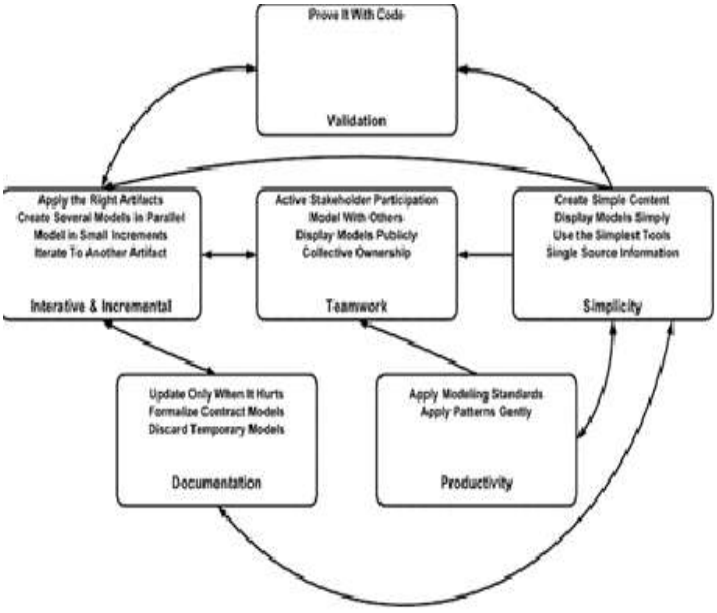
Instructor Notes:

Agile Modeling

- It is a variant of the Incremental model
- It enables developing customized software with a process that helps in meeting current requirement as well as future through suitable adjustment
- The following principles that enable this methodology to be effective and light weight
  - **Communication**
    - Open communication between stakeholder and development team at every stage
  - **Simplicity**
    - The model emphasize the need to keep concepts and ideas in simple manner like simple tools , simple design , content etc..
  - **Feedback**
    - Model allows quick feedback from shareholders to ensure that things are on track
- Typically used when requirements are volatile and applications are time critical and the team is aware of the agile practices

Agile Modeling enables developers to develop a customized software development process that actually meets their current development needs and is flexible enough to adjust in the future.

Agile Model Characteristics:
Active stakeholder involvement
Collective ownership
Keep it simple (content , design , tools)
Model in small increments
Apply modelling standards



Instructor Notes:

Software Operations and Maintenance

- Software maintenance in Software Engineering is the process of modifying a software product after delivery
- The modification of s/w could be for various reasons
  - Fixing a defect - Corrective
  - Address incremental and performance Improvements Perfective
  - Perfecting and adapting the code to the changes in operating environment - Adaptive
- Maintenance activity over the years has evolved to become a crucial source of input and a key driver for new product requirements
- Software maintenance and support projects also follow a life cycle model
  - Quick Fix Model
  - Code reuse Model
  - Iterative enhancement Model, etc..

Software Maintenance and support has over the years evolved into a crucial phase and also supports life cycle model . Some of the models are discussed in brief

- **Quick-Fix Model :** This model is adhoc and has a “firefighting approach” . That is fix the bug quickly when it occurs
- **Bohem’s Model :** Here the list of “approved changes” are identified by management using various cost effective strategies , which are then executed based on the budgets and resources .
- **Iterative Enhancement Model :** Originally proposed for developmental model later extended to maintenance as well . Follows the same iterative methods as that of development. Implementation of the changes to the system are done in an iterative way throughout the life of the system
- **Reuse Oriented Model :** This model is based on the principle that maintenance is an activity based on reuse of existing components Has the following activity
  - Identify parts of the old system which are candidates for reuse
  - Understand the items
  - Modify the items based on new requirements
  - Integrate the modified parts into the new system

Instructor Notes:

Software Maintenance Life Cycle – Typical phases

- Application Assessment
  - Understand the application and client expectation
  - Prepare Project Plan
- Knowledge Transition/Responsibility Transition
  - Ramp up the team
  - Sign off service level agreement
- Steady State – Maintenance Release (MR) execution
  - Provide maintenance support
  - Provide production support
  - Monitor Performance

Application Assessment : Here the application is thoroughly studied in terms of functionality, quality, volatility , workflows, user satisfaction (As-is) , along with customer future goals and expectation . Based on the assessment a high level project plan is prepared .

Knowledge Transition : Here a detailed plan is drawn to transit all the necessary information from earlier vendor of the customer including Documents , lessons learnt , service status and expectation etc. Transition from earlier vendor is desirable , if not possible both new vendor and the customer work together to ensure that the new vendor is effective enough to take over the projects/services etc . A detailed plan is drawn to ramp up the resources .

Execution : Here application maintenance team and production support team perform tasks to ensure the application is up and running without any defects . Requirements are consolidated , analyzed , estimated , coded , tested and deployed as per the *SLA* , against MR (Maintenance Request) .

Instructor Notes:

Selection of life cycle and support

- Based on nature of project
  - Clarity of requirements
  - Priority of implementation
  - Need to address change management
  - Need for prototypes
- Organization Support
  - QMS Procedure /guidelines for performing various activities
  - Templates/ forms/checklist/metrics for tracking , measuring and analysing various activities and taking corrective action
  - Tools for automating and improvement
    - SVN /TFS (for CM) (Recommended)

Note : In some cases we may use tools/templates suggested by clients

Instructor Notes:

Instructor Notes:

What is a Requirement?

- Simply put , it is the needs of the stakeholder which needs to be met/satisfied (by a s/w)
- A Software capability needed by the User to solve a problem to achieve an objective.
- Can be a high level abstract statement indicating needs to a details of the system which the client can validate
- Requirements needs to be
  - Elicited
  - Analyzed
  - Specified
  - Managed
- The engineering process covering all activities leading to discovery , document and manage requirement is known as Requirement Engineering

A requirement is a capability or condition to which the system must conform. Software requirements provide a “black box” definition of the system. They define only those externally observable “What’s” of the system, not the “How’s.”

Requirements are very important for any project, or sub-section of a project, because they define what will be built, hence requires a rigorous engineering process, , hence the term Requirement engineering .

Requirement engineering is a continuous activity throughout the lifetime of a software as requirements are subject to change . New requirements needs to be elucidated existing requirements revamped etc.

Instructor Notes:

Requirement phase

- This is the initial phase of the development process
- The development team works closely with the customer to determine the customer's requirements for the product – functional, non functional and other characteristics which the product must mandatorily have .
- The requirements identified in this phase serve as a foundation for the remaining phases of the development process, and the customer acceptance criteria.
- The main participants involved in the requirement phase are
  - Stake holders
  - Requirement Engineer

Stakeholders are individuals who affect or are affected by the software product
They have some influence over the software , in terms of requirements
Stakeholders can be categorized as

Acquirers of the software (both management and users)
Suppliers of the software (individuals and team , management)
Others (Sales , Legal teams , other internal teams)

RE who are also known as requirements engineer, business analyst, system analyst, product manager, or simply analyst
RA's primary responsibility is to gather, analyze, document and validate the needs of the project stakeholders.They help to determine the difference between what customers say they want and what they really need

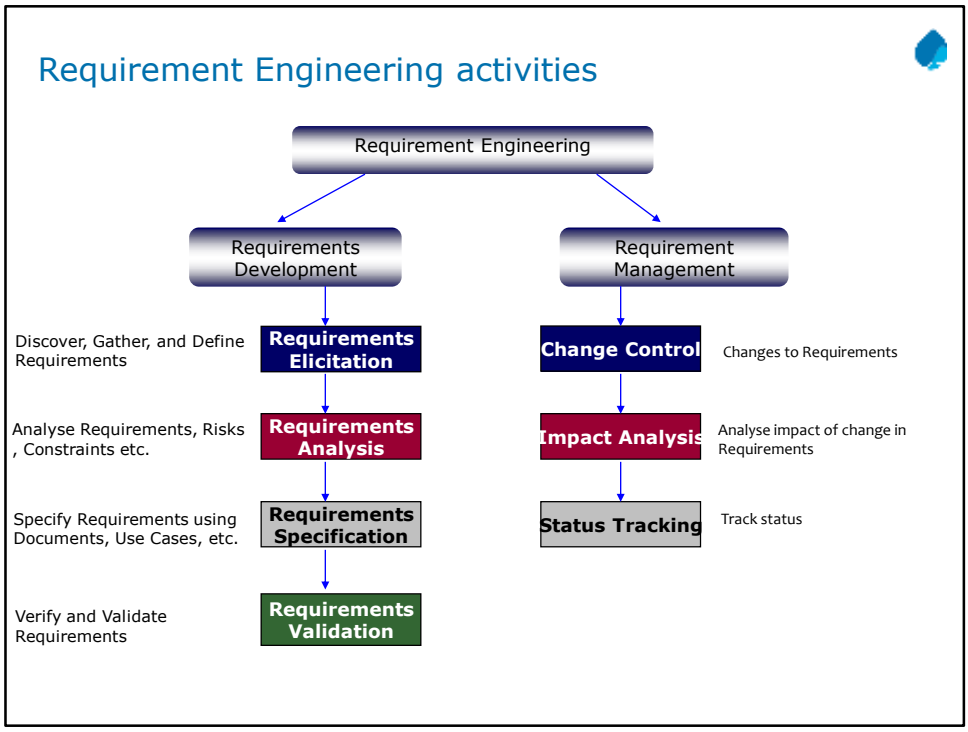
Identifying and considering the needs of all of the different stakeholders can help prevent requirements from being overlooked.
Requirement s can be classified under two categories :
Functional : Requirements what the system should do or provide for users .They can include all the business processes /funcionality, reports and queries and details of data to be stored and managed .
Non Functional : Non-functional requirements are constraints, targets or control mechanisms for the new system. They describe how, how well the system should e provide services like response time , ease of use-usability , security, recoverability etc.

Instructor Notes:

Need for good requirements

- Requirement Problems are the single No.1 reason for projects failing over
  - Schedule
  - Budget
  - Scope
  - Quality
  - And even getting Cancelled!!
- Reworking requirements cost 40-50% of project effort
- Many problems found during design, testing, or operation of a system are the result of incorrect, incomplete, or missing requirements

Instructor Notes:



Requirements Engineering = Requirements Development + Requirements Management

Instructor Notes:

Requirement Engineering activities

- Requirement Elicitation
  - This phase focuses on examining and gathering desired requirements and objectives for the system from different stakeholders
  - Various techniques are followed to gather requirements viz interviews, document examining , brainstorming , prototyping etc
- Requirement Analysis
  - This phase focusses on analyzing rigorously ,classifying, prioritizing , documenting the gathered requirements within business context
- Requirement Specification and Validation
  - A formal document is prepared after collating all requirements which contains a complete description of the external behavior of the software system.
  - Requirements are specified in
    - URS User Requirement Specification
    - SRS System Requirement Specification
    - Use Case Documentation
  - **T**he requirement documented in the SRS is verified , validated and agreed upon by all parties .

Requirements Specifications include

- What is in scope and out of scope
- Related or referenced documents (Customer supplied artifacts and materials)
- Requirement providers and stakeholders of the project
- Deliverables & delivery dates
- Risks and assumptions
- Current and proposed business system
- Acceptance criteria and Customer CTQs
- Functional and non functional requirements
- Limitations and constraints

URS : User Requirement Specification
Typically written prior to the SRS, based on the user's experience and expectations, with inputs from stakeholders

SRS : System Requirement Specification
This information includes detailed descriptions of the operations performed by each screen, the data that can be entered into the system , work-flows performed by the system and system reports or other outputs,
. An SRS also specifies who can enter data into the system as well as how the system meets regulatory requirements that are applicable to the specific system.

Use Case Documents : The document and diagrams together forms the UCD .
Typically done when the approach is Use case modelling

QMS provides templates for creating specification document

Instructor Notes:

Requirement Validation and Management

- Requirement Management
  - Requirements Management (RM) involves recognizing and planning changes occurring in requirements due to various factors during the life of the project
  - RM is a continuous activity that can occur post development during maintenance as requirements may continuously change
  - When a sizeable set of changes are received, the project may decide to go thru a change request process , to get approval for time and budget

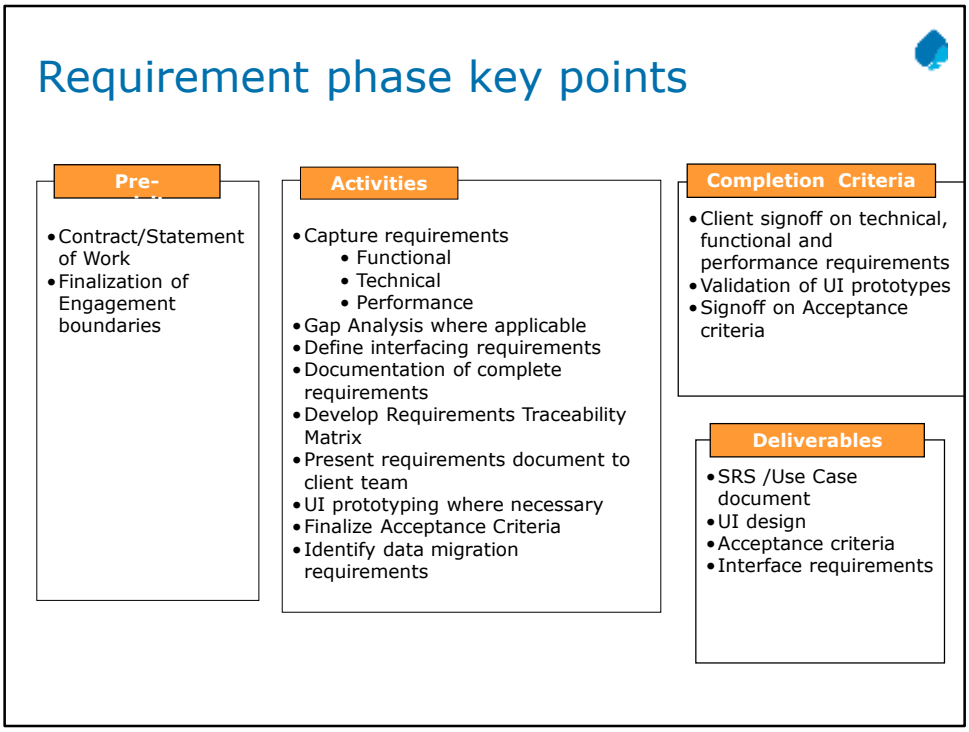
RM phase controls and tracks the changes of agreed requirements, relationships between requirements, and dependencies between the various produced during software engineering process

Requirement may change due to various reasons

- A bug
- Technology change
- Change in business

When sizable requirement changes are received the changes are incorporated via a change management process .

Instructor Notes:



Instructor Notes:

Instructor Notes:

Architecture and Design

- Architecture
  - It is the high level organizing structure of the system
  - It defines the components, interfaces, and behaviors of the system.
  - The process of architecting a software involves defining a structured solution that meets all of the technical and operational requirements, along with attributes such as performance, security, and manageability.
  - This phase usually involves the technical/solution architect
- Design
  - It is a process of creating a detailed specification for a software module .
  - It involves algorithmic design and other implementation specific approaches for a s/w component such as modularity , control hierarchy, data structures etc
  - Designers /Technical leads ,senior developers , architects are involved in this phase

Architecture deals with Non functional requirements whereas design deals with functional

The architecture of a system is its 'skeleton'. It's the highest level of abstraction of a system. What kind of data storage is present, how do modules interact with each other, what recovery systems are in place.
Software design is about designing the individual modules / components. What are the responsibilities, functions, of module x? Of class Y? What can it do, and what not? What design patterns can be used?
So in short, Software architecture is more about the design of the entire system, while software design emphasizes on module / component / class level

Instructor Notes:

Key activities in Design phase

- The design phase includes following activities
  - Identify solution which will meet the customers non functional requirements like performance , security etc..
  - Identify technology stack
  - Identify framework and design pattern
  - Create software architectural overview document
  - Identify major modules and its interfacing with each other as well as external systems if any
  - Defining the logical and physical database model
  - Create test design
  - Plan of the unit and integration test cases
  - Detailing the overall logic of the module in pseudo code or flow charts
  - Detailed database design including constraints data types etc.. (Physical)
  - Detailed interfacing reference (with API and parameters
  - Prepare design documents

Architecture constitutes of the following key activities:

- Solution space for “non-functional requirements”
- Decision on Technology Stack
- Framework requirements definition and solution
- Critical decisions for some risky "functional" requirements

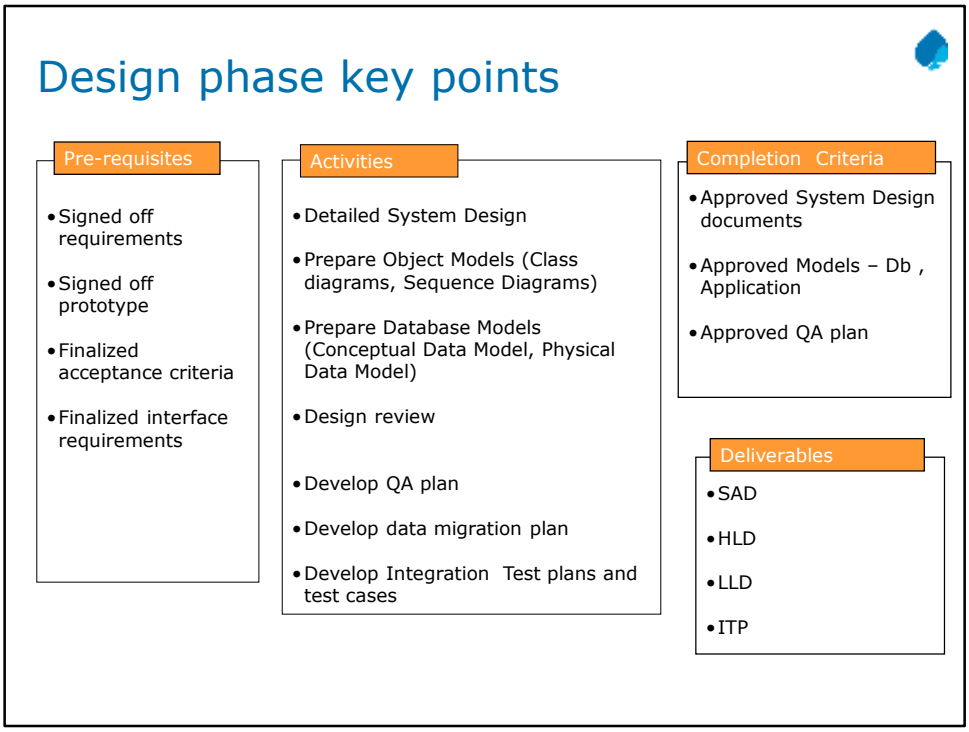
Architecture activities are delivered by the Technical Architect and supported by the Design lead **Design** is mainly focused on modeling the functional aspects of an application.
Solution space for “functional requirements” based on defined architecture
Design Pattern choice
Application design
Logical ER Data Model (entities, attributes, relationships)
UML Models - Class, Sequence , Activity etc
Analysis Model (domain entities, control and boundary classes, their functional attributes and associations)
Additional UML diagrams (as needed)
Data types of attributes
Additional classes, attributes for technical implementation (ex. primary key)

Design activities are delivered by the Design Lead and the Designer .Design Lead is a key role and which acts as a communicator between the architect and designers

Instructor Notes:

Instructor Notes

Show a sample design document



Instructor Notes:

Instructor Notes:

Construction phase

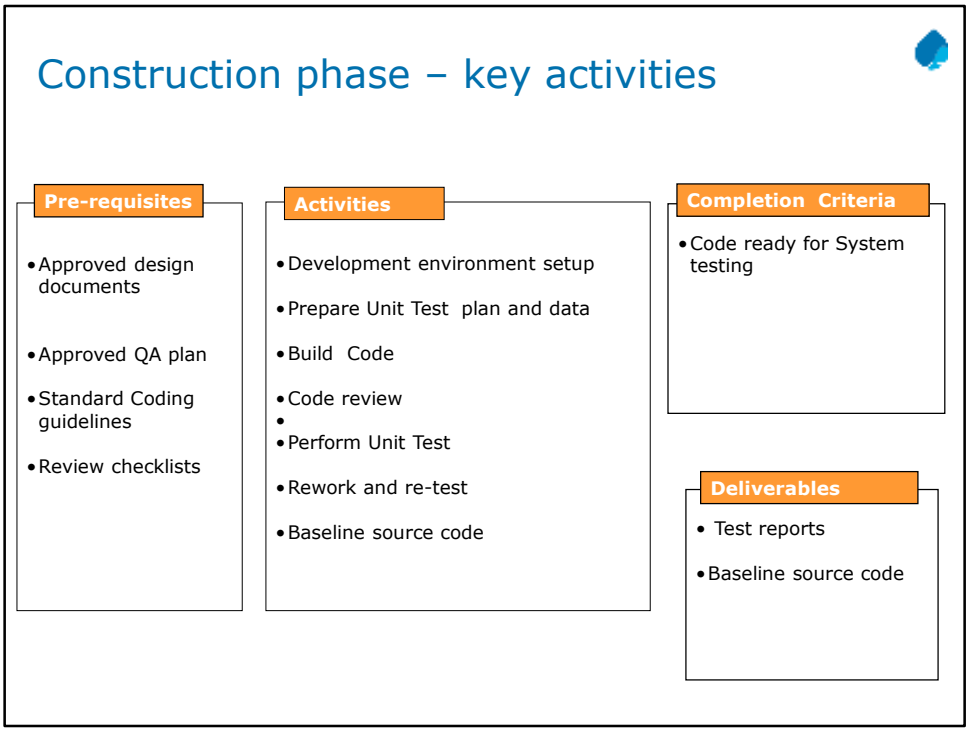
- Also known as implementation phase
- Main objective of this phase is to translate the software design into code , each component identified in design is implemented as a program module following coding guidelines
- Each module in this phase is reviewed and unit tested to determine correct working (White Box testing)
- Unit tested code are then integrated in a planned and a phased manner .
- In each integration step the partially integrated system is tested

Instructor Notes:

Construction phase

- In addition to the major activities the following activities are also carried out as well
  - Prepare unit test plan and test case
  - Prepare unit test data
  - Setup coding guidelines
  - Setup the environment for Configuration Management as per CM guidelines
  - Provide suitable environment for base lining code and continuous integration
  - Defect reporting and fixing
- The main role players in this phase are
  - Developers
  - Team Leads

Instructor Notes:



Instructor Notes:

Instructor Notes:

System Testing

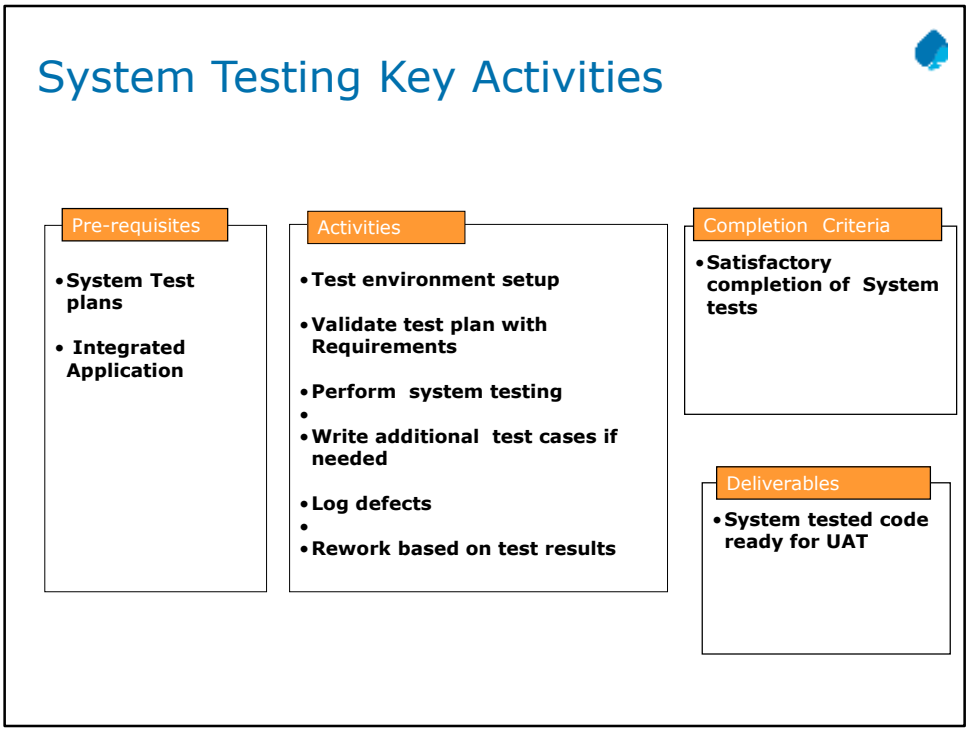
- System testing involves testing of all subsystems together
- Also known as Black Box testing It is ideally done by the QA team
- The following types of testing are done as part of system testing
  - Functional testing to validate functional requirements
  - Performance testing to validate non functional requirements

Goal of functional testing is to test the functionality of the system . The test cases are written from the requirement documents by the QA team as a parallel activity once the requirements are frozen . The system is treated as a black box (implementation independent) .

Goal of the performance testing is to validate the non functional requirement of the system (captured during requirements) , In this kind of testing the system is pushed to its limits to see how it behaves . Some of the performance tests

- **Stress testing** to test stress limits of system (maximum # of users, peak demands etc)
- **Volume testing** to test large volume of data
- **Security Testing** to test if the system behavior on security violation
- **Recovery Testing** to test system's response to loss of data and presence of errors
- **Usability testing** to test the ease of Use of the system

Instructor Notes:

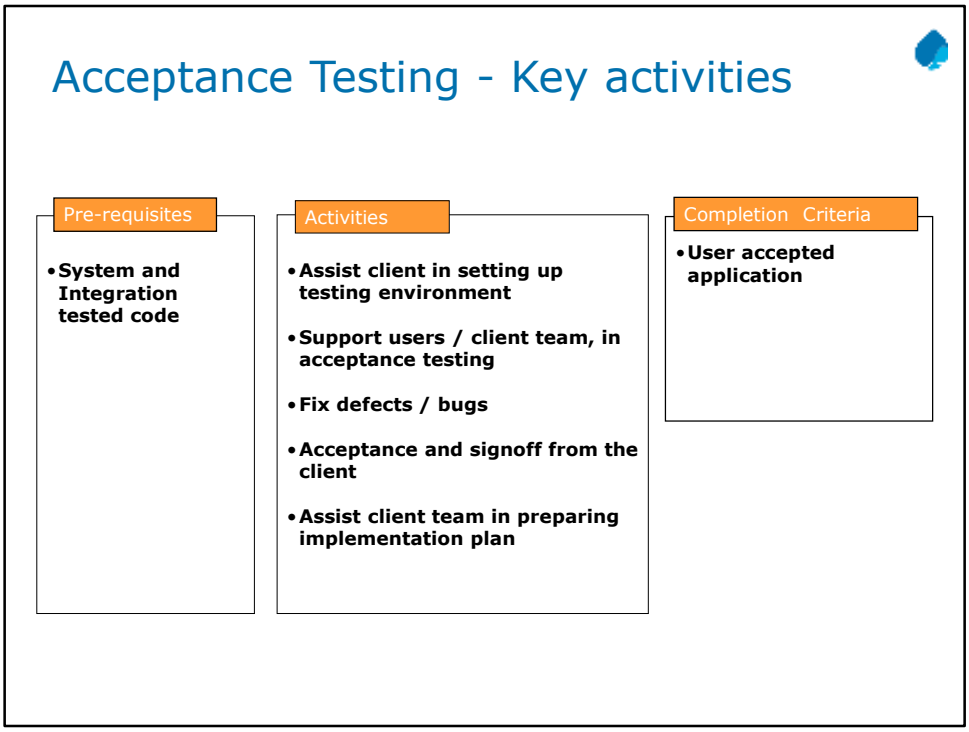


Instructor Notes:

Acceptance Testing

- Usually done at the client location by the client , after the findings of System testing is fixed
- Focus of Acceptance test is to evaluate the system's compliance with the business requirements and assess readiness for delivery.
- Acceptance Testing is done in two ways
  - Alpha Testing or Internal Acceptance Testing
    - **done by s/w vendors**
  - Beta Testing or User Acceptance testing
    - **Done by end users of customers or customer's customer**
- Outcome of the acceptance testing will enable the user, customers or other authorized entity to determine whether or not to accept the system.

Instructor Notes:



Instructor Notes:

Post Acceptance phase

- After successful acceptance testing plans are made to move the application to the "live environment"
- Activities like knowledge transfer , end user training , project signoff are also done .
- Once when the customers starts using the developed system the maintenance team supports and monitors the system to resolve errors and performance .

Instructor Notes:

Reviews and CM process are processes which will be used in all the phases of Software development . These are umbrella process .

Instructor Notes:

Reviews – What

- An assessment of a work product created during the software engineering process
- Ensure completeness and consistency of the work product
- Identify needed improvements
- It is a Quality Assurance mechanism to identify discrepancy /deviation from the accepted standards
- Goal of Review
  - To detect and eliminate defects early, effectively and before delivering the product to the customer

Instructor Notes:

Reviews – Why

- Intermediate software products which are not testable as standalone units
- Can find errors not possible through testing
  - E.g., Maintainability: Comments, Consistency, Standards
- Are proactive measure to find out 60-80 % of defects
- **Reduce Rework Effort and Improve Schedule adherence**
- Enables Quantitative Quality Assessment of any work product

Instructor Notes:

Software Reviews – When , where

- Can happen in all phases of SDLC
- All artifacts can go for reviews
  - Proposals, contracts, statement of work
  - All project work products - Plans, Configuration Mgmt, Test Plans
  - Deliverable and non deliverable work products
  - Software(e.g.: source code) and non software work products (e.g.: documents, test data, etc.)
- Process descriptions
- Policies, brochures, reports, guidelines, standards, training material where required

Instructor Notes:

Types of Review

- Self Review
  - Done by the author himself with the aid of tools like checklists , review guidelines , rules etc..
- Peer Review
  - Done by "peer" or colleague formally or informally using various approaches
    - Inspection
    - Walk through
    - Pair Programming

Inspection – It is a more systematic and rigorous type of peer review. Inspections are more effective at finding defects than are informal reviews. In inspection reviewer drives the review process .

Walkthrough – It is an informal review because the work product's author describes it to some colleagues and asks for suggestions. Walkthroughs are informal because they typically do not follow a defined procedure, do not specify exit criteria, require no management reporting, and generate no metrics.

Pair Programming – In Pair Programming, two developers work together on the same program at a single workstation and continuously reviewing their work.

Instructor Notes:

Instructor Notes:

Show the coding checklist which is there in QMS as sample .

Review Process

- Input
  - Work Product , Specifications, Checklists, Guidelines, Historical Data
- Process
  - Prepare for Review
  - Conduct Reviews
  - Analyze Deviations
  - Correct Defects
- Output
  - Review Form, reviewed work product,

Instructor Notes:

Instructor Notes:

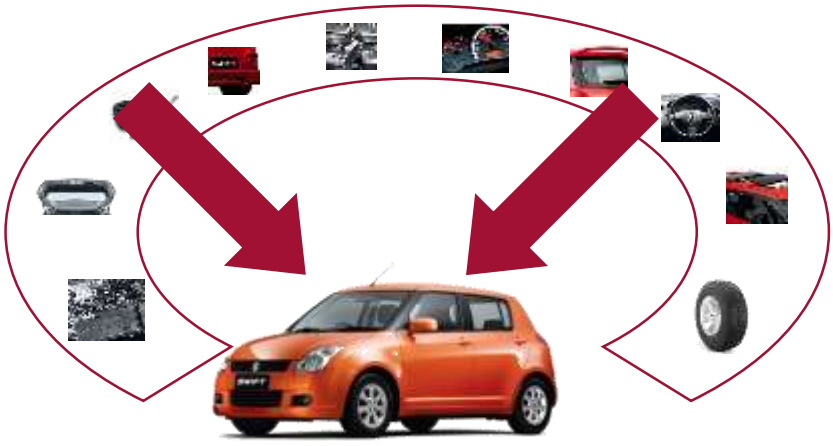
Agenda

- What is SCM?
- Why SCM?
- Elements of SCM
- Change Management
- Information on SCM tools

Instructor Notes:

What is a "Configuration"?

- Arrangement of functional unit of a system in a particular order



A configuration is an arrangement of functional units according to their nature, number, and chief characteristics. Often, configuration pertains to the choice of hardware, software, firmware, and documentation. The configuration affects system function and performance.

Instructor Notes:

What is Software Configuration Management?

- SCM is the overall management of a software project as it evolves into a software system.
- This includes managing , tracking, organizing, communicating, controlling modifications made in project including release plan
- Also includes the ability to control and manage change in a software project
- Configuration Managers Are responsible for planning the CM activities of their project
- The configuration details of the project are documented in the CMP (Configuration management Plan)

Instructor Notes:

Why do we need SCM?

- Some of the frustrating problems we face are
  - The latest version of the source code not found
  - A developed and tested feature is mysteriously missing
  - A fully tested program suddenly does not work
  - A wrong version of code was tested
- SCM answers who, what, when and why
  - Who makes the changes?
  - What changes were made to the system?
  - When were the changes made?
  - Why were the changes made?

SCM is the process that defines how to control and manage change.
The need for an SCM process is acutely felt when there are many developers and many versions of the software. Suffice to say that in a complex scenario where bug fixing should happen on multiple production systems and enhancements must be continued on the main code base, SCM acts as the backbone which can make this happen.

Without configuration Management the following can happen

- Unorganized project items
- Confused naming conventions
- Review / Delivery of wrong version of code
- Development based on old version of specifications
- No proper access / privilege control; Unauthorized access to secure information
- Redundant file creation
- Change Management becomes ineffective

Instructor Notes:

Elements of SCM

- Configuration identification
  - CI
  - NCI
- Configuration control (Elements)
  - Library Control
  - Access Control
  - Version Control
  - Establish Naming conventions
  - Establish Baselines
  - Branching, Merging and Labeling
- Change Management
- Auditing (Verification)

Instructor Notes:

Elements of SCM

- Configurable item (CI)
  - CI is a collection of items, treated as a unit which are likely to undergo change during the project life cycle and a change to them is likely to affect other CIs.
  - Items that needs to be accessed, controlled, secured and archived is a configurable item

    (E.g.) Design document, project plan etc..

- Non Configurable item (NCI)
  - Any item / file for which changes need NOT be tracked) i.e. no need to roll back to earlier versions is called a Non-Configured Item.

  (E.g.) Minutes of Meeting(MOM)

Version:

The term 'version' is used to define a stage in the evolution of a CI, for example versions of source code, etc.

Instructor Notes:

Library Structure

- Controlled collection of software and related documentation designed to aid in
  - software development
  - use
  - Maintenance

Organized structure

- The folder structure is indicated in the CMP

Instructor Notes:

Example of Libraries Structure

- Input Library
- Development Library
- Testing / Review Library
- Release / Delivery Library
- Template Library
- Project Management Library

Tip: The library (folder) can be created on need basis for the project. No thumb rule to create the same.

SDLC  **Folders**

Instructor Notes:

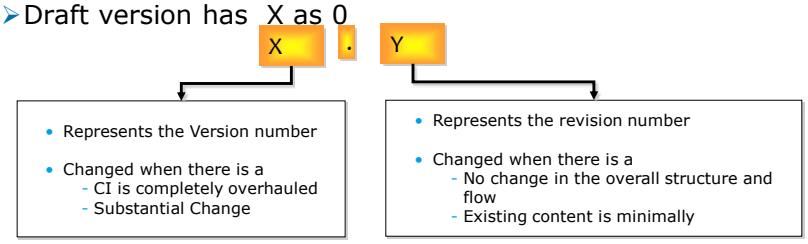
Usage of library - example

- Coding and Testing scenario
  - Development done in Development library by development team who have access to development folder
  - QA team (testing team) would be doing the testing
  - As per CM policy QA team wont have permission on Development folder ,
  - The code is **moved** from development folder to testing folder
  - The code **is moved back** to development folder for rework
  - The Re-testing happens in Testing library following the above steps
  - Once all the bugs are fixed , the code is moved to release folder .

Instructor Notes:

Version Numbering

- A version number is a unique number or set of numbers assigned to a specific release of a software/hardware/firmware
- As updates and new editions of product are released, the version number will increase
- Version numbers are usually divided into sets of numbers, separated by decimal points
- Draft version has X as 0



Instructor Notes:

Naming Conventions

- Helps in easy identification
- The name may include
  - Project name/ Application name/ Request ID
  - Document/ Work product name
  - Version number/ Date (If manual configuration)
  - Status – Draft, review, approval etc.,

Sample Document	Sample Naming conventions
Management documents	Est_<Project name>_dd_mm_yy_<ver_ x.y>.doc, PP_<Project name>_<ver_x.y>.doc, SCH_<Project Name>_<ver_x.y>.xls/mpp
Source code	<Component name>_<ver_ x.y>.java, <Module name>_<component name>_<ver_ x.y>.java
Unit Test Plan	UTP_<Component name>_<ver_ x.y>.doc
Quality Records	C-rev_<component name>_<ver_ x.y>**.xls, C-rev_<Module name>_<component name>_<ver_ x.y>**.xls

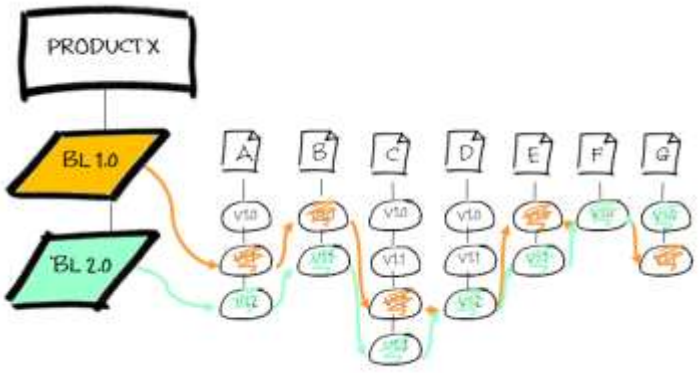
Instructor Notes:

Baselines

- Baseline
  - A 'baseline' is a CI that has been reviewed and agreed upon and is a basis for further development.
  - After base-lining all changes to CI are controlled through a formal change process (Such as Change Management and Configuration Control).
  - For example a reviewed and approved Project plan is used as a basis for execution of the project.
  - One baseline may have several work product , each having different version number
  - Baselining can be of many types – Input baseline , design baseline , code baseline etc

Instructor Notes:

Illustration of a Baseline



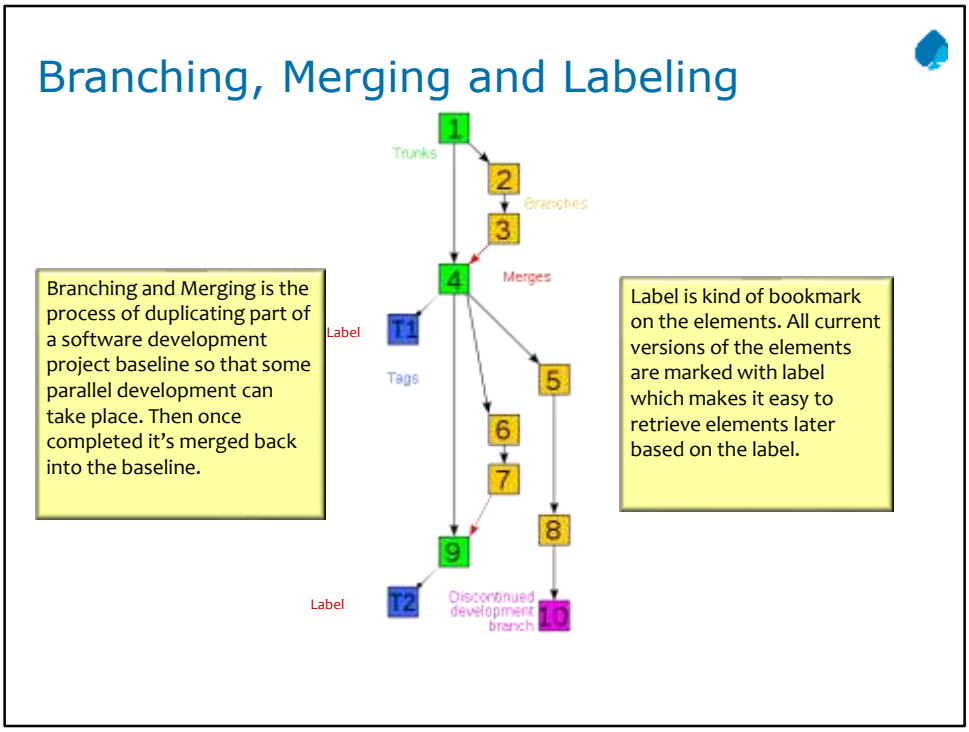
A baseline defines a set of files, each at a particular version. These need not be the latest (most recent) version. A baseline label uniquely identifies the configuration. Files may belong to one or more baselines.

In the example of Figure 12 baseline BL1.0 is the first baseline recorded. It consists of seven artefacts, each at a unique revision number. For this example, assume that BL1 records the most recent versions of each artefact. As development progresses each artefact is modified as required (that is, some artefact are modified, some are not). At some time later another baseline is taken – BL2.0. In this case BL2.0 records the current latest revisions of each file. Notice that artefact F is unchanged, so F v1.0 is included in both baseline BL1.0 and BL2.0.
In general each successive baseline contains more recent versions of files (but not always).

Instructor Notes:

Instructor Notes

This would be shown in the demo session of sv/tfs



Branching and Merging are two important aspects of version control. This concepts are extremely useful in parallel software development The two concepts are briefly explained below

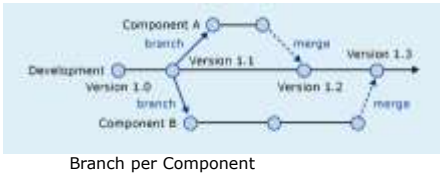
Branch : It is a line of development that exists independently of another line, yet still shares a common history. For example assume we are developing a banking application for American customers. The same application can be used by Canadians with some customization. T he solution to this requirement can be achieved by creating a branch for the Canadian customers and incorporating the needed changes. Since the two branches are related , if any changes /bug fixes needed in both can be easily duplicated . The main line of development is called **trunk (shown in the diagram), whereas a branch is a side line of a development**

Merge : In simple terminologies a merge is basically "copying" the changes across branches . To quote an example , assume that we have started working on the next version of our product (version 4.0) . A critical bug and some minor customization is asked for . To accommodate we create a branch to incorporate change and deploy to the customer. Once the next release is ready we merge the branch completely so as to incorporate the changes done in the branch in the new version

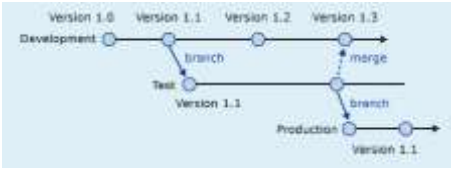
Instructor Notes:

Different Ways of Branching

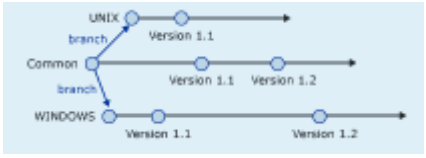
Branch per Release



Branch per Component



Branch per Promotion



Branch per Technology

Branch per Release
Every release is a new branch; common changes are merged between the releases. Branches are killed off only when the releases are no longer supported.

Branch per Component
Each architectural component of the system is a new, independent branch. Components are merged into the main branch as they are completed.

Branch per Promotion

Every tier is a permanent branch. As changes are completed and tested, they pass the quality gate and are "promoted" as merges into successive tiers.

Branch per Technology
Each technology platform is a permanent branch. Common parts of the codebase are merged between each platform.

Instructor Notes:

SCM Tools

- Tools help in managing projects better as it reduces a lot of manual effort
- Early SCM tools had facilities for controlling and managing CI , but now it comes with a lot of features like
  - Build and Release management
  - Defect Management and tracking
  - Packaging control etc ..
- Major advantages of SCM tools are
  - Sharing of project information across teams with accurate and reliable information,
  - Flexible in supporting parallel development
  - Provide better decision-making capability by managing and tracking

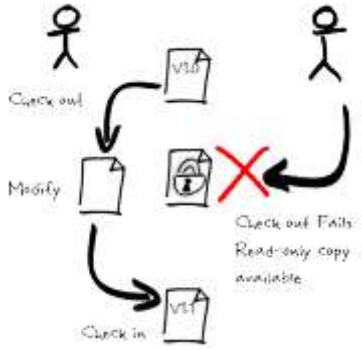
Instructor Notes:

Different Roles and Accesses in SCM Tool

- Administrator
  - All access to all project folders
  - Can add users, create and manage projects and modify their access levels
- Configuration Manager
  - Greater access than all team-members.
  - Creates the basic environment for his projects configuration management
  - Responsible for moving files across projects, establishing baselines, adding requirements files, preparing guidelines, etc
- Team-member
  - Varying access depending on their responsibilities. For e.g. PM gets add/modify access to Management project

Instructor Notes:

Check –in and Check -Out



A check-out is the act of creating a local working copy from the repository. A user may specify a specific revision or obtain the latest.

A check - in is the action of writing or merging the changes made in the working copy back to the repository.

File locking
In a file locking system only one developer has write access to the artifact Other developers will have read-only access to the current (stored) version. The file is only available again once it is checked back in.

Instructor Notes:

Test your understanding

- SRS (System Requirement Specification) are prepared in which phase ?
- A software can be built using more than 1 life cycle model . (T/F)
- Name some non functional requirements
- Architecture focuses on functional requirement T/F ?
- Alpha testing and beta testing happens in which phase ?

Instructor Notes:

Test Your Understanding!

- In which phase UNIT Test Plan is written ?
- White box testing includes
  - Unit Testing
  - System Testing
  - Operations Acceptance Testing
  - Integration testing
- A single baseline may contain many files.(T/F)
- A Tester can test in the development library.(T/F)